Overview

BAD_Mutations (**B**LAST-**A**ligned-**D**eleterious?) performs a likelihood ratio test (LRT) for the prediction of deleterious variants. The package is comprised of Python and Bourne Again Shell (BASH) scripts. The LRT is handled by a HYPHY script. BAD_Mutations was written with Python 2 syntax, but conversion to Python 3 is planned. BAD_Mutations is designed to be run from the command line. Running from an interactive Python environment is not recommended nor supported.

BAD_Mutations contains five major subcommands: `setup`, `fetch`, `align`, `predict`, and `compile`. Both `setup` and `fetch` are meant to be run once, or very rarely. The `align` subcommand generates phylogenetic trees and multiple sequence alignments for input to the prediction scripts. The `predict` subcommand does the actual variant effect prediction. More information about how to run BAD_Mutations is available in the "Usage" section.

Briefly, BAD_Mutations predicts deleterious variants using a sequence constraint approach. For a given query gene sequence and list of nonsynonmyous SNPs, a multiple sequence alignment among orthologues is produced, and the given codons are tested for conservation. Variants that alter a codon with a high degree of conservation are inferred to be deleterious. More details on the procedure in BAD_Mutations is available in the "Methods" section.

Citation

The model used to estimate codon conservation and predict which variants are deleterious is reported in Chun and Fay (2009). The actual software package is first used in Kono *et al.* (In Prep.). BAD_Mutations will have a formal publication after the Kono *et al.* manuscript is published.

BAD_Mutations was primarily written by Thomas JY Kono and Paul J Hoffman. The HYPHY script for estimating codon conservation was written by Justin C Fay. Testing was performed by Chaochih Liu, Felipe Reyes, and Skylar Wyant.

Downloading

BAD_Mutations is distributed through a GitHub repository. You can use Git to clone the repository, or download a ZIP archive from GitHub.

Dependencies

BAD_Mutations is written to run in a UNIX-like environment. It has been successfully run on both Apple OS X and GNU/Linux. It is not supported on Microsoft Windows. It has not been tested on other variants of commercial UNIX.

BAD_Mutations requires that the following software is installed and available in your $PATH or sys.path in Python:

- GNU Bash $\geq$ 3.2
- Python $\geq$ 2.6.x
- Biopython 1.6x
- argparse (Python library) If using Python 2.6
- BLAST+ $\geq$ 2.2.29
- PASTA
- HyPhy 2.2.x
- cURL

Note that if you plan to run many analyses in parallel, you should use a **single-threaded** version of HyPhy.

Instructions for UMN MSI

This section is specific to using BAD_Mutations on the University of Minnesoa Super Computing Institue cluser. Our cluster uses the module command to add and remove certain programs from the user's environment. The following commands should be run for BAD_Mutations on the cluster:

```
$ module load python2
$ module load biopython
$ module load ncbi_blast+
$ module load hyphy/2.2.6_smp
```

You will have to install PASTA as its user manual instructs. cURL should be available on MSI.

Input

Input files should be plain text with UNIX line endings (LF). BAD_Mutations takes a FASTA file containing the query coding sequence, and a text file with the list of codons to

predict. The coding sequence does not have to start with ATG, but it should be supplied in the 5′ to 3′ direction, and its length should be a multiple of 3. The codons should be supplied as numerical offsets with respect to the provided FASTA file, with counting starting from 1 and one codon per line. The substitutions file may optionally have a second field with a SNP identifier.

There is no programmatic means of enforcing the consistency of directionality between the FASTA file and the substitutions file. This means it is possible to submit them in the reverse order, but keep in mind that the coordinates must match in order for the predictions to be valid.

The FASTA input should look like this:

```
>Gene_1
ATGCCAGTGCAG...
...
```

And the substitutions file should look like this:

```
4   SNP_1
10  SNP_2
25  SNP_3
100 SNP_4
```

This pair of files would describe four nonsynonymous variants to predict in a single coding sequence. The variants occur at residue numbers 4, 10, 25, and 100 in the **amino acid** sequence, with the first residue being treated as position 1. Their identifiers are `SNP_1`, `SNP_2`, `SNP_3`, and `SNP_4`, respectively. These may be any non-whitespace text, and may be internal identifiers for bookkeeping, or rs numbers, or some other SNP identification system.

Note that while the FASTA file contains **nucleotide** sequence, the substitutions file contains positions in the **amino acid** sequence. Support for nucleotide offsets is planned for a future version.

For compiling the raw HyPhy outputs (one per gene) into a final report, you must also supply an effects table as generated by SNP_Effect_Predictor.py. This table is required, as part of the significance testing involves polarizing nonsynonymous SNPs by their ancestral states, and this information is not present in the raw HyPhy output.

Output

BAD_Mutations returns a report with information for each queried position. Information in the report includes the number of species in the alignment, the alignment column for the queried codon, a constraint score, a p-value associated with the LRT, and a constraint score and p-value with the query sequence masked from the alignment to reduce reference bias. Information is also available in the multiple sequence alignment, phylogenetic tree, and raw HyPhy output, which are all kept as intermediate files.

Usage

Basic Invocation

BAD_Mutations can be called from command line in a manner similar to UNIX programs. You must either set the executable flag on the script BAD_Mutations.py, or pass the script to the Python interpreter.

```
$ chmod +x BAD_Mutations.py
$ ./BAD_Mutations.py [Options] [Subcommand] [More Options ... ]
--OR--
$ python BAD_Mutations.py [Options] [Subcommand] [More Options ... ]
```

BAD_Mutations offers five subcommands, setup, fetch, align, predict, and compile. They are summarized below. As of the current version, setup and compile are not fully implemented.

Subcommands, Options, and Switches

Note: BAD_Mutations example command lines will be provided at the end of the setup, predict and fetch sections below.

General Options

BAD_Mutations takes the following general options:

Option	Value	Description
-h	NA	Show help message and exit.
-v/--verbose	'DEBUG'	Be very verbose. Print all messages.
	'INFO'	Just print info, warning, and error messages. Useful for progress checking.
	'WARNING'	Print warnings and errors. Default setting.
	'ERROR'	Only print error messages.
	'CRITICAL'	Print almost nothing. Critical failures only.

The `setup` Subcommand

The `setup` subcommand creates a configuration file that contains paths to required executables, paths to data storage directories, BLAST search parameters, alignment parameters, and prediction parameters. Running `setup` is optional, but recommended as it makes standardizing across genes and analyses much simpler. This subcommand can also download and compile dependencies for `BAD_Mutations` .

NOTE: This subcommand is currently being developed. The function prototypes are present, but they currently do not work.

The `setup` subcommand takes the following options:

Option	Value	Description
`--list-species`	NA	Show all species databases available.
`-c/--config`	[FILE]	Where to store the configuration file. Defaults to `LRTPredict_Config.txt`.
`-b/--base`	[DIR]	Directory to store the BLAST databases. Defaults to the current directory.
`-d/--deps-dir`	[DIR]	Directory to download and store the dependencies. Defaults to current directory.
`-t/--target`	[SP_NAME]	Target species name. Must be one of the species (case sensitive) given by `--list-species`. This species will be excluded from the prediction pipeline to avoid reference bias. No default.
`-e/--evalue`	[FLOAT]	E-value threshold for accepting TBLASTX hits as putative orthologues. Defaults to 0.05.
`-m/--missing`	[INT]	Minimum number of gapped (missing) sites in the multiple species alignment (MSA) to be considered for prediction.

The `fetch` Subcommand

The `fetch` subcommand creates the necessary BLAST databases for identifying orthologues. It will fetch gzipped CDS FASTA files from both Phytozome 10 and Ensembl Plants, unzip them, and convert them into BLAST databases. Fetching data from Phytozome requires a (free) account with the JGI Genome Portal. Note that not every genome sequence in Phytozome is available to be used for this analysis. Check the species info page on Phytozome for specific data usage policies.

The `fetch` subcommand accepts the following options:

Option	Value	Description
-c/--config	[FILE]	Path to configuration file. Defaults to LRTPredict_Config.txt.
-b/--base*	[DIR]	Directory to store the BLAST databases. Defaults to the current directory.
-u/--user	[STR]	Username for JGI Genome Portal. Required.
-p/--password	[STR]	Password for JGI Genome Portal. If not supplied on command line, will prompt user for the password.
--fetch-only	NA	If supplied, do not convert CDS FASTA files into BLAST databases.
--convert-only	NA	If supplied, only unzip and convert FASTA files into BLAST databases. Do not download.

*: If this value is supplied on the command line, it will override the value set in the configuration file.

The `align` Subcommand

The `align` subcommand will run BLAST to identify putative orthologues against each species' CDS sequence database. The putative orthologues are aligned with PASTA, and a phylogenetic tree is estimated from the alignment.

The `align` subcommand accepts the following options:

Option	Value	Description
-b/--base*	[DIR]	Directory to store the BLAST databases. Defaults to the current directory.
-c/--config	[FILE]	Path to configuration file. Defaults to LRTPredict_Config.txt.
-e/--evalue*	[FLOAT]	E-value threshold for accepting TBLASTX hits as putative orthologues. Defaults to 0.05.
-f/--fasta	[FILE]	Path to FASTA file with query sequence. Required.
-o/--output	[DIR]	Directory for output. Defaults to current directory.

*: If this value is supplied on the command line, it will override the value set in the configuration file.

The `predict` Subcommand

The `predict` subcommand will generate predictions for a list of affected codons. It will run a BLAST search of the query sequence against each CDS sequence that was downloaded with the `fetch` subcommand, pick the likely homologous sequences, align them, and then use HyPhy to predict each query codon.

The `predict` subcommand accepts the following options:

Option	Value	Description
`-a/--alignment`	[FILE]	Path to the multiple sequence alignment file. Required.
`-c/--config`	[FILE]	Path to configuration file. Defaults to `LRTPredict_Config.txt`.
`-r/--tree`	[FILE]	Path to the phylogenetic tree. Required.
`-s/--substitutions`	[FILE]	Path to substitutions file. Required
`-o/--output`	[DIR]	Directory for output. Defaults to current directory.

*: If this value is supplied on the command line, it will override the value set in the configuration file.

The `compile` Subcommand

The `compile` subcommand will take an output directory containing HyPhy output files, and produce a table with predictions for each variant. The script will print P-values, but will not assess significance, as a suitable significance threshold cannot be determined programmatically. This is left to the user to interpret. This subcommand requires the output from another SNP effect script, SNP_Effect_Predictor.py (NOTE: requires the companion Python class defined in gff_parse.py).

The `compile` subcommand accepts the following options:

Option	Value	Description
-S/--long-subs	[FILE]	Path to the SNP effect table. Required.
-p/--pred-dir	[DIR]	Output directory from the predict subcommand. Required.

Example Command Lines

The following command line demonstrates the typical usage of BAD_Mutations .

This command will set up the environment for predicting in barley (*Hordeum vulgare*), with very verbose output:

```
$ ./BAD_Mutations.py -v DEBUG \
                     setup \
                     -b /scratch/BAD_Mutations_Data \
                     -d /scratch/BAD_Mutations_Deps \
                     -t 'Hordeum_vulgare' \
                     -e 0.05 \
                     -m 0.2 \
                     -c BAD_Mutations_Config.txt 2> Setup.log
```

This command will download all of the necessary CDS sequences from both Phytozome and Ensembl Plants and convert them into BLAST databases:

```
$ ./BAD_Mutations.py -v DEBUG \
                     fetch \
                     -c BAD_Mutations_Config.txt \
                     -u 'user@domain.com' \
                     -p 'ReallyGoodPassword123' 2> Fetch.log
```

And this command will predict the functional impact of variants listed in subs.txt using CoolGene.fasta as a query:

```
$ ./BAD_Mutations.py -v DEBUG \
                     predict \
                     -c BAD_Mutations_Config.txt \
                     -f CoolGene.fasta \
                     -s subs.txt 2> CoolGene_Predictions.log
```

A Note on Parallel Execution

BAD_Mutations is designed to run all predictions in a single thread. There is a joke in here somewhere about Python programs and lack of concurrency . . . For now, all functions and supporting scripts are written for single-thread execution, and parallel execution can be done with a tool like GNU Parallel. Native parallel support is planned for a future release.

Configuration File Format

NOTE: The configuration file format is under revision (in a new git branch) and is planned to change soon. This section of the manual will be updated when the new file format is deployed. The format will follow the specifications used by the Python ConfigParser module.

The configuration file is modeled after the configuration file of STRUCTURE [Pritchard *et al.*, (2000)]. A sample configuration file is shown below:

```
// Generated by 'setup' at 2015-10-07 19:09:09.622228
#define BASE /scratch/BAD_Mutations_Data
#define TARGET_SPECIES hordeum_vulgare
#define EVAL_THRESHOLD 0.05
#define MISSING_THRESHOLD 10

// Program paths
#define BASH /usr/local/bin/bash
#define GZIP /usr/bin/gzip
#define SUM /usr/bin/sum
#define TBLASTX /usr/local/bin/tblastx
#define PASTA /usr/local/bin/run_pasta.py
#define HYPHY /usr/local/bin/HYPHYSP
```

Runtimes and Benchmarks

By far, the slowest part of BAD_Mutations is fetching CDS sequences and converting them to BLAST databases. This may take up to several hours, depending on your network and disk speeds. The databases and FASTA files take up approximately 4GB, as of October 2015. As more genomes are sequenced and annotated, this figure will increase.

For a typical barley gene (≈3000 bp), BAD_Mutations can generate a phylogenetic tree and multiple sequence alignment in approximately 5-10 minutes on a desktop computer

(Intel i7 2.8GHz). Note, however, that not every gene will have every species represented in the alignment and tree. This is not a problem for BAD_Mutations .

Predictions are generated in two stages: a $\frac{dN}{dS}$ estimation phase and a per-site prediction phase. The $\frac{dN}{dS}$ phase is slow; for the same ≈3000bp gene, the average time to estimate $\frac{dN}{dS}$ is 11319.5 CPU-seconds (≈3 CPU-hours), with a standard deviation of 10803.9 CPU-seconds (also ≈3 CPU-hours). Per-site predictions are much faster, with an average runtime of 73.9 CPU-seconds, and a standard deviation of 67.8 CPU-seconds.

In all, BLAST searching and predicting for a single barley gene takes an average of 3-4 CPU-hours to complete. The process is readily parallelizable on a gene-by-gene basis. This makes processing a complete dataset consisting of tens of thousands of genes feasible on a computing cluster.

Note, however, that runtimes will vary depending on the gene being analyzed. Genes that are rapidly evolving will take longer in the BLAST search, alignment, and prediction stages. The max amount of time it took for BAD_Mutations to calcuate $\frac{dN}{dS}$ was ≈46 CPU-hours, for instance.

Methods

BAD_Mutations uses TBLASTX to identify genes that are homologous to the query sequence based on translated similarity. Hits that are above the user-supplied E-value threshold are excluded. Once a list of orthlogues is identified, BAD_Mutations translates the sequences into amino acids, and aligns them with PASTA. A phylogenetic tree of the species is also estimated from the alignment. The alignment is then back-translated using the original nucleotide sequence hits from their respective BLAST databases. This alignment is then supplied to the prediction script, where the query codons are evaluated using HyPhy.

Evaluation of codons uses a likelihood ratio test (LRT) to give the probability that a nonsynonymous SNP is deleterious. First, the ratio of the local synonymous and nonsynonymous substitution rates ($\frac{dN}{dS}$) is estimated from the gene alignment. Then, using those rates and the estimated phylogenetic relationship among the sequences, the program tests the likelihood of the queried codon evolving under selective constraint against the likelihood of it evolving neutrally. For a full description of the statistical model used, see Chun and Fay (2009).

BAD_Mutations makes several assumptions in its prediction pipeline. First, putative orthologues identified with BLAST are assumed to have conserved function across all of the species represented in the alignment. For some gene families, particularly those involved in pathogen recognition and defense, this assumption may not be true. Next, BAD_Mutations assumes that the sequences identified as homologous through sequence similarity are *orthologous*. This assumption is manifest in the multiple sequence align-

ment, as each site in the alignment is then assumed to be orthologous. For gene families that are highly duplicated (either proliferating, or due to a whole genome duplication event), this assumption may also be violated. That is, sequences identified through BLAST searching may be paralogous, and subject to a different mode of selection than purifying selection.

As such, exercise caution when interpreting results from BAD_Mutations .

Data Sources

As of October 2015, the following Angiosperm genomes (41) are available for use in Ensembl and Phytozome:

Species	Common Name	Assembly Version	Annotation Version	Source
Aegilops tauschii	Goatgrass	ASM34733v1	1	Ensembl Plants
Aquilegia coerulea	Columbine	1.1	1.1	Phytozome 10
Arabidopsis lyrata	Lyrate rockcress	1.0	1.0	Phytozome 10
Arabidopsis thaliana	Thale cress	TAIR10	TAIR10	Phytozome 10
Boechera stricta	Drummond's rockcress	1.2	1.2	Phytozome 10
Brachypodium distachyon	Purple false brome	2.1	2.1	Phytozome 10
Brassica oleracea	Cabbage	2.1	2.1	Ensembl Plants
Brassica rapa	Turnip mustard	FPsc 1.3	1	Phytozome 10
Capsella grandiflora	–	1.1	1.1	Phytozome 10
Capsella rubella	Red shepherd's purse	1.0	1.0	Phytozome 10
Carica papaya	Papaya	ASGPBv0.4	ASGPBv0.4	Phytozome 10
Citrus clementina	Clementine	1.0	clementine1.0	Phytozome 10
Citrus sinensis	Sweet orange	1.0	orange1.1	Phytozome 10
Cucumis sativus	Cucumber	1.0	1.0	Phytozome 10
Eucalyptus grandis	Eucalyptus	2.0	2.0	Phytozome 10
Eutrema salsugineum	Salt cress	1.0	1.0	Phytozome 10
Fragaria vesca	Strawberry	1.1	1.1	Phytozome 10
Glycine max	Soybean	a2	a2.v1	Phytozome 10
Gossypium raimondii	Cotton	2.1	2.1	Phytozome 10
Hordeum vulgare	Barley	082214v1	1.0	Ensembl Plants
Leersia perrieri	Cutgrass	1.4	1.0	Ensembl Plants
Linum usitatissimum	Flax	1.0	1.0	Phytozome 10
Malus domestica	Apple	1.0	1.0	Phytozome 10
Manihot esculenta	Cassava	6.0	6.1	Phytozome 10
Medicago truncatula	Barrel medic	Mt4.0	Mt4.0v1	Phytozome 10
Mimulus guttatus	Monkey flower	2.0	2.0	Phytozome 10
Musa acuminata	Banana	MA1	MA1	Ensembl Plants
Oryza sativa	Asian rice	IRGSP-1.0	7.0	Phytozome 10
Panicum virgatum	Switchgrass	1.0	1.1	Phytozome 10
Phaseolus vulgaris	Common bean	1.0	1.0	Phytozome 10
Populus trichocarpa	Western poplar	3.0	3.0	Phytozome 10
Prunus persica	Peach	2.0	2.1	Phytozome 10
Ricinus communis	Castor bean	0.1	0.1	Phytozome 10
Setaria italica	Foxtail millet	2.0	2.1	Phytozome 10
Solanum lycopersicum	Tomato	SL2.50	iTAG2.3	Phytozome 10
Solanum tuberosum	Potato	3_2.1.10	3.4	Phytozome 10
Sorghum bicolor	Milo	2.0	2.1	Phytozome 10
Theobroma cacao	Cacao	1.0	1.0	Phytozome 10
Triticum urartu	Red wild einkorn	ASM34745v1	1	Ensembl Plants
Vitis vinifera	Grape	Genoscope.12X	Genoscope.12X	Phytozome 10
Zea mays	Maize	6a	6a	Phytozome 10